Validating Data Using SAS Formats as a Programming Tool

John R. Gerlach, Independent Consultant, Hamilton, NJ

Abstract

Proper data analysis requires well-defined data that have been validated. Typically, the transition from raw data to a viable SAS® data library requires more effort than just writing an INPUT statement. In fact, the quest for validated data often becomes a project in itself consuming tremendous resources. Considering that many validation checks can be written as Boolean expressions, “if gender not in (‘F’, ‘M’),” it’s possible that these expressions can be stored in a user-defined format.

Besides its typical use of creating formats for reading and writing data, the FORMAT procedure offers much as a programming tool, such as doing table look-ups and sub-setting data. And, with some ingenuity, this procedure can greatly facilitate the validation process.

This SAS paper discusses techniques for facilitating the validation process, focusing on a macro and a user-defined format catalog that together perform validation checks on a data library to produce validation reports. Equally important, the validation checks are maintained by simply modifying the format catalog; whereupon, the validation process is performed, as needed, by invoking the macro.

Introduction

The real world of data analysis tends to be rampant with convoluted specifications and unclean data. Even worse, the proliferation of SAS programs needed for any project often emulates an ad hoc analysis, rather than a well organized and documented system of SAS programs ensuring correct results that can be replicated, as in a clinical trial. Consequently, the project endures many hardships, complete with a nasty deadline, while attempting to clarify ambiguous specifications and to clean messy data.

Consider a clinical trial study consisting of several protocols representing hundreds of patients and lots of data, including: demographics, lab tests, vital signs, and dosing, each represented by its own SAS data set. It would be nice to have a data validation tool that can be used repeatedly, thereby facilitating data management and ensuring integrity of the data. Also, it would be even better if the validation checks were well organized rather than buried in a collection of programs. Assuming that the validation checks can be written as Boolean expressions, it is possible to store each check as the label portion of a numeric SAS format such that there would be \( n \)-levels in the Value statement of the FORMAT procedure. Then, using the format, along with the Macro Language and some Base SAS, the validation process becomes almost trivial and very efficient. In fact, for the end-user, the validation becomes a matter of maintaining a format library and invoking a SAS macro with several parameters, as needed.

A Quick Overview of the Format Procedure

A format in SAS is an instruction to the SAS System for the purpose of reading or writing a data value. SAS informats, so-called, are instructions to the SAS System on how to read data; whereas, SAS formats are instructions to write data. SAS provides a rich collection of standard informats and formats used for reading and writing character, numeric, and datetime values, several of which are listed below. The Format procedure creates two kinds of formats: value formats, which convert output values into a different form; and, picture formats that specify templates for printing numbers. Only numeric variables can have picture formats associated to them.
Several SAS Supplied Informats (Reading Data)  

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>w.</code></td>
<td>Standard numeric data</td>
</tr>
<tr>
<td><code>lbw.d</code></td>
<td>Integer binary data</td>
</tr>
<tr>
<td><code>OCTALw.d</code></td>
<td>Converts positive octal values to integers</td>
</tr>
<tr>
<td><code>PERCENTw.</code></td>
<td>Converts percentages to numeric values</td>
</tr>
<tr>
<td><code>DATEw.</code></td>
<td>Dates (DDMMYY)</td>
</tr>
<tr>
<td><code>DDMMYYw.</code></td>
<td>Date values</td>
</tr>
<tr>
<td><code>JULIANw.</code></td>
<td>Julian date (e.g., YYDDD)</td>
</tr>
<tr>
<td><code>MONYYw.</code></td>
<td>Month and year</td>
</tr>
<tr>
<td><code>$w.$</code></td>
<td>Standard character data</td>
</tr>
<tr>
<td><code>$CHARw.$</code></td>
<td>Character data with blanks</td>
</tr>
<tr>
<td><code>$EBCDICw.$</code></td>
<td>EBCDIC to native format</td>
</tr>
<tr>
<td><code>$VARYINGw.$</code></td>
<td>Varying-length char values</td>
</tr>
</tbody>
</table>

Several SAS Supplied Formats (Writing Data)  

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>w.</code></td>
<td>Standard numeric data</td>
</tr>
<tr>
<td><code>w.d</code></td>
<td>Standard numeric data</td>
</tr>
<tr>
<td><code>BESTw.</code></td>
<td>SAS chooses best notation</td>
</tr>
<tr>
<td><code>COMMAw.d</code></td>
<td>Commas in numbers</td>
</tr>
<tr>
<td><code>DOLLARw.d</code></td>
<td>Dollar sign &amp; commas</td>
</tr>
<tr>
<td><code>DATEw.</code></td>
<td>Dates (DDMMYY)</td>
</tr>
<tr>
<td><code>JULIANw.</code></td>
<td>Julian date (e.g., YYDDD)</td>
</tr>
<tr>
<td><code>MONYYw.</code></td>
<td>Month and year</td>
</tr>
<tr>
<td><code>$w.</code></td>
<td>Standard character data</td>
</tr>
<tr>
<td><code>$CHARw.$</code></td>
<td>Character data with blanks</td>
</tr>
<tr>
<td><code>$UPCASEw.$</code></td>
<td>Converts text to uppercase</td>
</tr>
<tr>
<td><code>$VARYINGw.$</code></td>
<td>Varying-length char values</td>
</tr>
</tbody>
</table>

Exercise #1: Run the following program to study formats with respect to actual (raw) data values. Also, consider how SAS formats compare with SAS functions.

```sas
data values;
  input integer 5. +1 octal octal3. +1 binnbr binary9. +1 pcent
     percent4.2 +1 datel date9. +1 date2 ddmmyy8. +1
     state $2. hexval;
  wkday1  = put(date1, weekday.);
  wkday2  = weekday(date1);
cards;
  10008 012 000000100 0.09 22MAY2005 22/05/05 AZ 15
;  
  proc print;
    format integer comma5. octal 5. binnbr 5. pcent percent6.1
       datel date9. date2 ddmmyy8. state $2. hexval hex2.;
    title1 'Comparison of Internal and Formatted Values';
  run;
```

Besides having access to many standard SAS informats and formats, known collectively as SAS formats, you can create user-defined informats and formats using the Format procedure. User-defined informats convert input values to a different form for processing data; whereas, a user-defined format converts a value to a different form for output. Also, user-defined informats only read character (text) data. That is, such informats can convert character data into real numeric values; but, they cannot convert real numbers into characters. User-defined formats always associate variable values with character values. Thus, even formatted numeric values are character numbers, which explains why the SAS System writes a warning message when performing an arithmetic operation on a user-formatted numeric value.

User-defined formats are stored as entries in a SAS catalog, not as part of a SAS data set. Thus, permanently associating a user-defined format with a variable does not mean that they become a part of the data set. In fact, the format catalog need not reside even in the same data library as the data set. Thus, complications will arise if you do not have access to a user-defined format that has been associated with a data set variable.

Once the format exists, there are several ways to associate a format to a variable:

- Using an INPUT, PUT, ATTRIB, INFORMAT, or FORMAT statement in a DATA step
- Using an INFORMAT or FORMAT statement in a PROC step.
A Simple Validation Check

Assume that you have a typical SAS data set that contains both numeric and character data. What if you want to know whether any of the variables contained missing values? Certainly, you are not interested in all the possible values for each variable (especially for numeric data). Rather, you want to know the distribution of Missing versus Non-Missing values for every variable. Would you write a Data step?

Consider using the FREQ procedure with formats such that the values would indicate two levels only, that is, either Missing or Non-Missing values for each variable. Obviously, since the data set might contain both character and numeric variables, you would need two compatible formats. The user-defined formats below transform any variable into two-values: Missing or Non-Missing.

```
proc format;
    value missf  . = 'Missing    ';
                        other = 'Non-Missing';
    value $missf  ' ' = 'Missing    ';
                        other = 'Non-Missing';
run;
```

When applied to the FREQ procedure, the `missf` and `$missf` formats transform all the internal values of every variable into two levels only, as follows. Notice the use of the SAS automatic variables (e.g., `_all_`), which makes the names of the actual variables irrelevant.

```
proc freq;
    tables _all_ / missing;
    format _character_ $missf. _numeric_ missf.;
    title1 'Distribution of Missing / Non-Missing Values';
run;
```

Exercise #2:  Modify the above procedure so that the FREQ procedure executes inside a macro that has two arguments: the name of the SAS data set and the enumerated variables to be analyzed. Do we need to modify the FORMAT statement? No. Why not? Because SAS will use the FORMAT statement accordingly, whether you specify all the variables, only numeric variables, or only character variables.

Exercise #3:  Enhance the macro so that the second parameter, which denotes enumerated variables, has a default value of all numeric variables only.

```
%macro missrep(dsn, vars=_numeric_);
  proc freq data=&dsn. ;
  tables &vars. / missing;
  format _character_ $missf. _numeric_ missf.;
  title1 'Distribution of Missing / Non-Missing Values';
  run;
%mend missrep;
%missrep(study.demog, vars=age gender bdate);
```

The Control Input / Output Data Set

In order to understand the idea of storing validation checks into a SAS format catalog, it is necessary to take a look at Control data sets used by the FORMAT procedure. And, to facilitate our learning, let’s create a couple of formats the old-fashioned way, using the VALUE statement, as follows.
proc format cntlout=fmtlib;
value sexf 1 = 'Male'
    2 = 'Female'
    3 = 'Unknown';

value $sexf 'F' = 'Female'
    'M' = 'Male'
    'U' = 'Unknown';
run;

proc print;
title1 'Listing of Control Output Data Set';
run;

Exercise #4:  Run the code above and observe the report representing a Control Output data set.

Exercise #5:  Expand the format procedure to include two more formats: $race and trtgrpf (treatment group), then print out the control data set.  Notice that the control data set has more observations.

value $racef 'C' = 'Caucasian'
    'B' = 'Black'
    other = 'Other';

value trtgrpf 1 = 'Study Drug'
    2 = 'Placebo';

Storing Validation Checks as Formats

Below is a syntax diagram that illustrates how to store validation checks using the VALUE statement in the FORMAT procedure. Notice that the format is numeric and that the LABEL portion of the format contains only the Boolean expression, not the IF statement.

proc format;
    value Name of format containing QC checks
        1 = "Boolean expression"
        2 = "Boolean expression"
        :     :     :     :
        n = "Boolean expression";
run;

Below is an example of creating several formats that contain validation checks. The numeric formats qcpat, qclab, and qcdose contain a collection of checks specific to the several data sets PATIENTS, LABS, and DOSE, respectively. Notice that each Boolean expression must be reasonable and syntactically appropriate with respect to the variables in the data set. Otherwise, obviously, you would get syntax errors when invoking the %qcrep macro that performs the validation task.

Exercise #6:  Create three formats that contain the following validation checks. Notice that the validation checks imply a ‘negative’ response, that is, what the data should NOT manifest.

Demographic data (qdem) –
1. Gender is neither ‘M’ or ‘F’
2. Age is greater than 24 or age is missing
3. Weight (WT) is less than 35 pounds
4. Gender is ‘M’
Laboratory data (qclab) –
1. Systolic (SYST) is greater than 200 and diastolic is greater than 150
2. Either systolic or diastolic reading is missing

Dosing data (qc dose) –
1. Date is missing
2. Dose titration level is not 3, 6, 12, or 25.

```sas
proc format;
    value qcdem 1 = "gender not in('M', 'F')"
              2 = "age gt 24 | age eq ."
              3 = "wt le 35"
              4 = "gender eq 'M';"
    value qclab 1 = "syst gt 200 and dias gt 150"
                   2 = "syst eq . or dias eq ."
    value qc dose 1 = "date eq ."
                   2 = "dose not in(3,6,12,25)";
run;
```

With this arrangement, you can add or delete validation checks simply by modifying the format. Also, notice that the format library containing the validation checks and the data sets can reside in separate data libraries. Moreover, each format is an independent collection of Boolean expressions that are pertinent to a specific SAS data set used for the sole purpose of validating it.

**Exercise #7:** Produce a report listing the aforementioned formats and observe the output.

```sas
proc format fmtlib;
    select qcdem qclab qc dose;
    title1 'Listing of Validation Checks Stored in Format Library';
run;
```

Notice that the output shows numeric formats whose labels are Booleans expressions. Once the validation checks are stored in a user-defined numeric format, we’re ready to use the `%qcrep` macro in order to run the validation process.

**The `%qcrep` Macro**

The `%qcrep` macro performs the validation process. It contains five parameters, two of which are positional parameters that require user input while the remaining are keyword parameters that have default values and are not required, depending on the situation. The parameters are listed below.

- **qcdset** Specify the name of the temporary or permanent SAS data set being validated.
- **qcfmt** Specify the name of the user-defined numeric format that contains the validation checks.
- **qc lib** (optional) Specify the name of the SAS data library where the format containing the validation checks resides. The default is the WORK library. Also, the format need not reside in the same library as the data set being validated.
- **qc var** (optional) List the variables in the data set that are pertinent to the validation process. The default is all variables. This option is useful when you wish to limit the variables being processed, excluding those variables that are typically irrelevant to the validation process (e.g., patient’s name). This option affects the detailed listing, as well.
- **qcdet** (optional) Request a detailed listing of errors, Yes or No. The default is Yes; otherwise, you will receive only a report indicating the frequency of each error.
The following explains how the `%qcrep` macro performs the validation process:

1. The OPTIONS statement ensures that the output is paginated and dated.

2. The FORMAT procedure selects the format containing the validation checks and creates an output data set containing only the variable `start`, which is numeric ranging from 1 to `n`, denoting the number of checks.

3. A Data _NULL_ step proceeds to create the macro variables: `&qcchk1`, `&qcchk2`, …., `&qcchkn`, representing the validation checks and the macro variable `&qcnchk`, denoting the number of validation checks.

4. The next Data step analyzes the data set as specified by the `&qcdset` parameter, keeping only those variables of interest depending on the `&qcvar` parameter. The %DO loop generates sequential IF statements, each with a DO / END block, for every validation check. The macro variable `&&qcchk&i..` resolves to an IF expression, that is, the respective validation check. Consequently, the Data step will generate an observation whenever the expression is true (e.g., gender not in('M','F')). The result is the data set `qcrep` which contains all the variables of interest originating from the user-specified data set being analyzed and the variable `qcchk` which indicates the error in the data set.

5. The FREQ procedure creates an output data set containing the variables `qcchk` and `count` that represent the frequency of occurrence of each check. FREQ does not produce standard output because of its 16-byte limitation (prior to SAS Release 6.12). Subsequently, the PRINT procedure produces the first report using the very same format which contains the validation checks.

Assuming that the user wants a detailed report of the validation checks, the SORT and PRINT procedures generate the second report. Otherwise, the user obtains only a frequency report. In the event that the data set being validated contains no errors, the `%qcrep` macro generates no reports. Instead, the SAS log indicates that the reporting procedures inside the macro processed data sets having 0 observations. No run-time error occurs since the data sets exist, albeit empty; hence, the option NODSNFERR is not required. By design, the macro does not even check for this situation.

The following examples illustrate proper use of the `%qcrep` macro.

```
%qcrep(study01.patient, qcpat)
```

Validates the data set `patient` using the `qcpat` format, which resides in the WORK library because of the default value of the `qclib` parameter. Also, the macro uses all the variables in the patient data set and generates a detailed report, as well.

```
%qcrep(study01.lab, qclab, qclib=study02,qcvar=pid vdate syst dias, qcdet=N)
```

Validates the data set `lab` using the `qclab` format, which resides in a different library. Also, the macro uses only several variables from the data set and does not generate a detailed report.

```
%qcrep(study.dose, qcdose, qclib=study02,qcvar=pid date dose, qcdet=Y)
```

Validates the data set `dose` using the `qcdose` format, which resides in a different library. Also, the macro uses only several variables from the data set and generates a detailed report.

**Exercise #8:** Run the following macro invocations and observe the output.

```
%qcrep(study01.patient, qcpat)
%qcrep(study01.lab, qclab, qclib=study02, qcvar=pid vdate syst dias, qcdet=N)
```
Using an Existing QC Format

Suppose you wish to validate a data set that contains exclusion criteria in a clinical trial study and you learn of an existing format that contains most of the forty plus validation checks of interest. However, you need to exclude some of the checks and add new ones. And, of course, you do not have access to the original program that created the original format.

Using the FORMAT procedure and a subsequent Data step, you can clone an existing format to meet your specific needs. Consider the following code that clones an existing format from one protocol for the purpose of validating a dose file belonging to another protocol in the same study.

```sas
proc format library=prot01 cntlout=qcexclu(keep=fmtname type start label) fmtlib;
  select qcexclu;
run;

data qcexclu;
  set qcexclu(where=(left(start) not in('1', '3', '18', '23')));
  start = put(_n_, 8.);
run;
proc format library=prot02 cntlin=qcexclu;
run;
```

The CNTLOUT option above creates a control data set that contains the information you need to revise the format. Also, the FMTLIB option produces a report so that you can determine which validation checks to keep.

Upon inspection of the format, you decide to exclude checks 1,3,18, and 23. The Data step modifies the control data set accordingly. Notice the LEFT and PUT functions. The variable `start` in the control data set, which denotes the \(i\)th validation check, contains right justified character data having a length of 16 bytes. Thus, the LEFT function properly subsets the data set, that is, it deletes those unwanted validation checks. Then, the PUT function assigns new ordered values (1,2, \ldots, n) to the variable `start` using SAS automatic variable `_N_`. Finally, the FORMAT procedures uses the control data set and creates a new format that resides in a different data library, as shown below. Adding validation checks to an existing format requires more effort.

Alternatively, the validation checks could be stored in a data set, even maintained by a front-end. Then, using a Data step and the CNTLIN option of the FORMAT procedure, the format can be created during the validation process.

**Exercise #9:** Using the QC format for dosing criteria (`qcdose`), exclude the first two validation checks, thereby creating a new QC format called `qcdosx`.

**A Large Table Look-up**

Imagine that you need to validation a collection of lab tests for a large international study. In this case, we need to create a format that serves as a table look-up. Because there are so many lab tests, it is not feasible to use the VALUE statement in the FORMAT procedure.

Fortunately, there is another source, a ‘bridge’ file that maps all unique verbatim terms to a code value based on the Physician’s Current Procedure Terminology (CPT) manual, from which we extract only the CPT codes of interest. An excerpt of the file follows.

<table>
<thead>
<tr>
<th>CPT4 Class Code / Verbatim Terms</th>
<th>CPT4 Class Code / Verbatim Terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>78350  *MISC – BONEDENSI</td>
<td>:</td>
</tr>
<tr>
<td>78350  BENE DENSITY SCREENING</td>
<td>:</td>
</tr>
<tr>
<td>78350  *X-RAY - BONE DENSITOMETRY</td>
<td>78350  BMD/DRU</td>
</tr>
<tr>
<td>78350  *X-RAY - BONE DENSITY</td>
<td>78350  BMD/PR</td>
</tr>
</tbody>
</table>
In order to create the appropriate format, we first read the file containing the CPT codes of interest, then create a user-defined format that facilitates the look-up process, illustrated below.

```plaintext
filename labs 'r:\study01\labs.dat';
data labtstf;
    length start label $60;
    retain fmtname 'LABTSTF' type 'C' label;
    infile labs firstobs=3 dlm='05'x missover;
    input label $5. @7 start $&;
    keep fmtname start type label;
run;
```

The following DATA step creates an observation that will represent the OTHER level as needed in the VALUE statement. This ensures that the format will handle all possible values in the verbatim term. That is, when the format cannot convert the verbatim term to a respective CPT code, then the formatted value is blank. The subsequent DATA step concatenates the two data sets LABTSTF and OTHER, thereby creating the appropriate control data set.

```plaintext
data other;
    length start label $40;
    retain fmtname 'LABTSTF' type 'C';
    start=' '; label=' '; hlo = 'O';
run;

data labtstf;
    set labtstf other;
run;
```

Once the data set contains all pertinent information needed, the CNTLIN option of the FORMAT procedure processes the data set and creates the format called LABTSTF.

```plaintext
proc format cntlin=labtstf maxselen=60;
run;
```

Then, the validation problem becomes trivial in that the DATA step asks the question: Does the verbatim term match to a CPT code of interest? If not, then there’s a problem.

```plaintext
data lab_errors;
    set labtests;
    if put(labtest,$labtstf.) eq ''
       then output;
run;
```
Exercise #10: Discuss whether it is possible to employ the Boolean expression in the table look-up as part of the collection of validation checks that are stored in a format.

Recoding Data

In lieu of a standard value, such as normal temperature, a special character (*) supplants the actual value in order to save time. However, when doing the analysis, the special character must be converted into its true value, which poses a programming task that includes an IF statement in a DATA step. An easier solution utilizes an enhanced numeric format, as follows. Then, when reading the raw data, simply use the informat in the following INPUT statement, as follows.

```
proc format;
invalue normt '*' = 98.6
    other = _same_;
run;
data;
infile cohort;
input pid $9. bp $6. temp normt.;
run;
```

Dynamic Assignment

Another situation involves a marketing survey that contains three sections of questions such that the scaled responses have different meaning and order. For example, Section #1 lists the responses in the order: Extremely Often, Frequently, Occasionally, Rarely and Very Rarely; using the letters A through E, respectively. However, Section #2 lists the responses in reverse order; and, Section #3 lists the responses in a different order from the others.

```
proc format;
  value sectf 1 = '$sect1f'
             2 = '$sect2f'
             3 = '$sect3f';
invalue $sect1f  'A' = 'Extremely Often'   'B' = 'Often'
             'C' = 'Occasionally'      'D' = 'Rarely'
             'E' = 'Very Rarely';
invalue $sect2f  'A' = 'Very Rarely'       'B' = 'Rarely'
             'C' = 'Occasionally'      'D' = 'Often'
             'E' = 'Extremely Often';
invalue $sect3f  'A' = 'Occasionally'      'B' = 'Often'
             'C' = 'Rarely'            'D' = 'Extremely Often'
             'E' = 'Very Rarely';
run;
```

Here the user-defined character informats $sect1f, $sect2f, $sect3f and the numeric format sectf provide the necessary information to convert the response data, having values A through E, according to its section in the survey.
Exercise #11: Submit the code above that creates dynamic format, then uses these formats. Afterwards, produce a frequency distribution of the variable RESP, denoting response.

Using Formats As Labels

When working with longitudinal data, assume you wish to print the actual dates for a specific time period and different text depending on other time periods. Using a standard SAS date format as a label and quoted text for the other ranges achieves this task. Consider the following SAS code that uses an existing SAS date format as a label denoting a specified time period, and descriptive text otherwise.

```sas
proc format;
  value titraf '01jan2004’d – ‘31mar2004’d = ‘Screening’
                 ‘01apr2004’d – ‘30sep2004’d = [date7.]
                 ‘01oct2004’d – ‘31mar2004’d = ‘Post Study’
                           other = ‘Error’;
run;
```

Exercise #12: Produce a listing of the dosing data using the `titraf` format.

```sas
proc print data=dose(obs=20);
  var patid date dose unit;
  format date titraf.;
  title1 ‘Listing of Dosing Data Set’;
run;
```

Conclusion

Whether your data represent a clinical trial intending to prove the efficacy of a drug or an analysis attempting to discern trends in the stock market, proper data analysis requires validated data. The initial goal of attaining good data must be done efficiently and effectively; otherwise, the expense incurred could, in fact, jeopardize the project or the integrity of the analysis.

The `%qcrep` macro facilitates the validation process and helps to ensure the integrity of the data. The macro is easy to use and produces two very useful reports. Also, the user-defined format that contains the actual validation checks is de facto documentation of the validation process.

Author Information

John R. Gerlach
Independent Consultant
Hamilton, NJ  08619
215.284.2176

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies.
Appendix A: The `%qcrep` Macro

```sas
%macro qcrep(qcdset, qcfmt, qclib=work, qcvar=_all_, qcdet=Y);
  options number pageno=1 date;
  proc format library=&qclib.
    cntlout=qcchks(keep=start);
    select &qcfmt.;
  run;
  data _null_; 
    set qcchks nobs=n;
    if _n_ eq 1
      then call symput('qcnchks',
                      left(put(n,2.)));
    call symput('qcchk'||
                  trim(left(put(_n_,2.))),
                  left(put(_n_,&qcfmt..)));
  run;
  data qcrep;
    set &dsn.(keep=&var.);
    %do i = 1 %to &qcnchks.;
      if &&qcchk&i..
        then do; qcchk = &i.; output; end;
    %end;
    run;
  proc freq data=qcrep noprint;
    tables qcchk / out=qccnts;
  run;
  title1 "QC Analysis of %upcase(&qcdset.)";
  proc print data=qccnts noobs label;
    var qcchk count;
    format qcchk &qcfmt.. count comma8.;
    label qcchk = 'QC Check'
                  count = '#';
    title2 "Frequency of QC Checks";
  run;
  %if %upcase(&qcdet.) eq Y
    %then %do;
      proc sort data=qcrep;
        by qcchk;
      run;
      proc print data=qcrep label noobs;
        by qcchk;
        format qcchk &qcfmt..;
        title2 "Detailed Listing";
      run;
    %end;
  %mend qcrep;
```